Sonatype

2015 State of the Software Supply Chain Report:

HIDDEN SPEED BUMPS ON THE ROAD TO "CONTINUOUS"

Foreword by Gene Kim, Gareth Rushgrove, John Willis, Jez Humble, and Nigel Simpson

RESEARCH REPORT

TABLE OF CONTENTS

Foreword 3

Introduction 5

Why All Modern Software Development Relies on a Software Supply Chain 6

SUPPLIERS: Open Source Projects 7

- Public Repositories (The Warehouses) 8
- Choosing the Best Suppliers (Sourcing) 9

PARTS: Open Source Components 12

- Repository Management (Local Warehouses) 15

MANUFACTURERS: Assembled Software Development 19

- Technical Debt: Assembly Line Inefficiencies 21

FINISHED GOODS: Software Applications 22

- The Volume of Elective Re-work and Risk 22
- Software Bill of Materials 23
- Quality Controls: OWASP, PCI, FS-ISAC, U.S. Congress 23

Lessons Learned from Traditional Manufacturing Supply Chains 25

Automation: How To Improve Software Supply Chains 26

Appendix 28

- Figure 1: The Volume and Size of the Global Software Supply Chain
- Figure 2: Target Benchmarks for Software Supply Chain Practices - Quality Control
- Figure 3: Target Benchmarks for Software Supply Chain Practices - Efficient Distribution
- Figure 4: Analysis of Components Used within Applications
- Figure 5: Multiple Versions of Parts Often Downloaded by the Largest Development Teams
- Figure 6: Volume of Defective Parts Used
- Figure 7: Comparison of Impact of Supply Chain Complexity on Prius versus Volt
- Figure 8: Efficient Sourcing Practices By Manufacturers

FOREWORD

Gene Kim, Co-author of "The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win" and upcoming "DevOps Cookbook"

"Anyone who believes, as I do, that we can learn valuable lessons from manufacturing and supply chains on how to better manage technology work will love this report. To describe how we assemble and integrate open source software into the services we create, Sonatype uses the metaphor of the 'software supply chain.' This metaphor enables some startling revelations on how we should select the components we use and the downstream effects of the decisions we make.

"As the custodians of the Central Repository, the largest open source repository in the world, Sonatype has insights into how the largest software supply chains in the world are managed—they've analyzed how over 106,000 organizations use over 1 million open source software components, spanning billions of component downloads.

"Just as in manufacturing, the effective management of our supply chains will create winners and losers. This will impact the quality of the services we deliver to our customers, as well as our ability to secure and maintain those services."

"Just as in manufacturing, the effective management of our supply chains will create winners and losers."

Jez Humble, Vice President at Chef, co-author of *"Continuous Delivery and Lean Enterprise"*

"The use of open source software components has become pervasive in the enterprise -- and rightly so. This reuse reduces complexity and time-to-market while increasing the security and reliability of software when done right.

"However, the research shows us many organizations are not doing it right. The result is a profusion of complexity in production systems that leads to poor quality, unreliable services as well as providing a rich target for bad actors.

"This report provides vital insight into the state of our software supply chain and practical advice on how organizations can innovate at scale while optimizing quality and security."

"This report provides vital insight into the state of our software supply chain."

Nigel Simpson, Director of Enterprise Architecture, Fortune 100 Media & Entertainment Company

"This report draws parallels with traditional manufacturing supply chains, giving us a new way to look at how we build software. With this fresh perspective we can see the importance of effectively managing the software supply chain, especially when software engineering is becoming increasingly commoditized through the use of reusable, off-the-shelf, open source components.

"This report draws parallels with traditional manufacturing supply chains."

"With visibility into the supply chain, we've identified many ways to streamline development and reduce risk. Just as in the automotive industry, where use of flawed components such as airbags has downstream impact on vehicles spanning multiple brands, we discovered that standardization of flawed open source components can impact hundreds of applications. Understanding the composition of our mission-critical applications has never been more important."

John Willis, DevOps Days core Organizer and co-author of the upcoming "DevOps Cookbook"

"Deming was one of the original prophets of Supply Chain hygiene. It's great to see this subject being discussed related to the software industry and more importantly dealing with Open Source.

"Moreover, in a world that is vastly moving to containers and immutable infrastructure, the subject of Software Supply Chains is going to become of increased importance. "

"In a world that is vastly moving to containers and immutable infrastructure the subject of Software Supply Chains is going to become of increased importance."

Gareth Rushgrove,
Senior Software Engineer, Puppet Labs / Curator of "DevOps Weekly"

"It's easier than ever to build complex systems quickly using open source components downloaded from the Internet. But where does that software (and its dependencies) come from? How do you keep it up to date? Is it introducing a critical security flaw to your application? The move towards microservices and polyglot programming environments makes these issues even more pressing, and the number of third-party components has grown too large to manage in a non-systematic way.

"This report is required reading for anyone interested in large-scale systems engineering."

"This report introduces approaches that can be used to mitigate the risks of poor software supply chain management. Better still, the discussion focuses on techniques that have been successfully applied in other industries, along with hard numbers that show the size of the problem. This report is required reading for anyone interested in large-scale systems engineering."

INTRODUCTION

Organizations continually strive to produce quality software even faster and more efficiently. Over the past decade, software development practices have witnessed significant changes that have greatly improved velocity, agility, and innovation. Key among these is the reliance on open source or third party component "building blocks" freely available from a wide variety of sources. As a result, 80-90% of a typical application is made up of components used to quickly add valuable features without custom coding[1]. The use of open source as well as other build components has led to the creation of a software supply chain.

While open source components and software supply chains are delivering huge benefits in terms of speed, the free-for-all nature of component availability and consumption practices also have led to significant waste, lower quality, and increased drag often not recognized by CIOs, enterprise architects, DevOps leaders, and software development teams. To begin improving the software supply chains that feed our software development practices, organizations first need to recognize that they exist and see the inefficiencies that are often hidden from view.

As the steward of the Central Repository, Sonatype watches over the world's largest public open source repository every day. Our role as steward enables us to provide a detailed perspective on the suppliers, consumption volumes, distribution mechanisms, and quality attributes of open source components that flow across today's software supply chains.

Our study of Central's use by more than 106,000 software development organizations in 2014 astounded us—so much that we wanted to share our analysis in this first-ever report on the State of the Software Supply Chain. We discovered that current practices are silently sabotaging a software development organization's efforts to accelerate development, improve efficiency and maintain quality. In short, our dependence on open source software components is growing faster than our ability to effectively source, manage, and secure it.

As you read this report, you'll discover eye-opening statistics on the usage of open source components. While this report offers sobering news about the state of the software supply chain, it also adds perspective on how a number of organizations have taken steps to improve their software supply chains. We need not reinvent the wheel. We need to recognize that traditional supply chain principles also apply to software development and can have the same transformative effect.

Given this evidence and the drive towards continuous and DevOps practices, software supply chain automation should be the new aim for software development organizations that want to make the world's best software, responsibly and profitably. This aim is simple and achievable. There are lessons that can be learned from traditional manufacturing supply chains to help organizations overcome the challenges identified through this research:

1. Use fewer and better suppliers
2. Use only the highest quality parts
3. Track what is used and where

We have seen enterprises apply these principles and boost developer productivity from 15 - 40% by:

- Reducing unplanned, unscheduled rework
- Reducing the mean-time-to-remediate issues
- Reducing the number of known defective parts
- Reducing mountains of technical debt
- Reducing complexity that leads to maintainability issues

WHY ALL MODERN SOFTWARE DEVELOPMENT RELIES ON A SOFTWARE SUPPLY CHAIN

A View Across 106,000 Organizations

In traditional manufacturing, we have suppliers, parts, warehouses, manufacturers, assembly lines and finished goods. In software development we have uncanny similarities. Yet software supply chains lack the rigor and processes that have fueled high-velocity production, operational efficiencies, and competitive differentiation witnessed in other supply chains.

If your company develops software, you're likely consuming thousands of open source and proprietary components. Although your aim is to produce the highest-quality software in the most efficient way, a closer look at the statistics shows a potentially different story. By understanding the software supply chain operating—largely hidden—at the core of your operations, you can more easily make small changes that yield dramatic gains.

As the steward of the Central Repository, the largest public open source repository, Sonatype has amassed a vast amount of data on the size, scale, velocity, and patterns of open source consumption across common software supply chains.

The Modern Software Supply Chain

You've got ...	You've got ...	You've got ...	You've got ...
SUPPLIERS Open Source Projects	**PARTS** Open Source Components & Warehouses	**MANUFACTURERS** Software Development Teams	**FINISHED GOODS** Software Applications

SUPPLIERS: OPEN SOURCE PROJECTS

A massive community of open source projects represents the external supplier base that feeds software supply chains. But unlike physical parts being consumed within traditional supply chains, once a software component is made available, it is never taken out of circulation. This means developers inadvertently can continue to use the old, outdated parts even when newer, better versions are available.

Open source means open access. It is community driven and community supported. Open source is absolutely essential for today's development processes. The increasing reliance on open source components has paralleled massive growth in component contributions from open source and third-party projects across all development languages. Where only a handful of open source projects were active in the early 1990's, OpenHub.net is currently tracking over 668,000 open source projects and over 3.7 million open source contributors[2]. The massive demand for open source components within software supply chains is being fed by this growing community of contributors and projects.

One significant difference between traditional suppliers and open source suppliers is that, in the open source world, components never reach their "end of life" or get retired. Once released into public repositories, outdated components continue to be available to software supply chains.

Module Counts (Components)

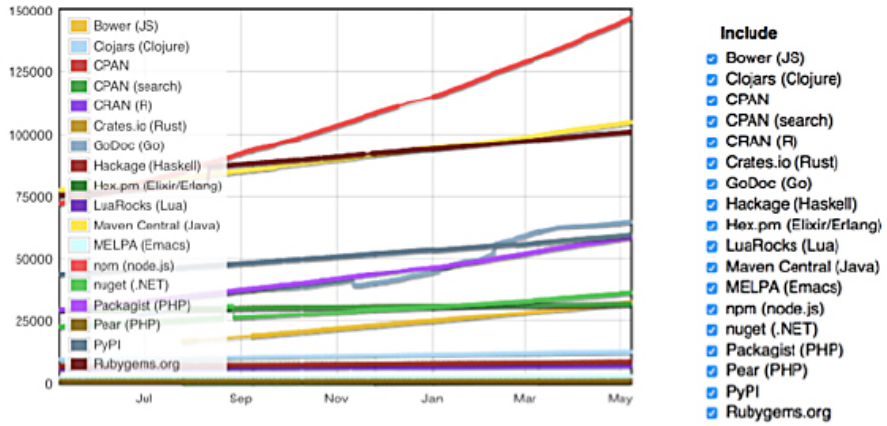


Image 3: Growth in new modules created by open source projects, 2014-2015.
Source: http://www.modulecounts.com

SUPPLIERS
Open Source Projects

DID YOU KNOW?

Nearly 1,000 new or updated components are added to the Central Repository every day.[5]

On average, components are updated 3.5 times per year. There is no way to inform development teams.[9]

Mean-time-to-repair a security vulnerability in component dependencies is 390 days.[10]

Public Repositories (The Warehouses)

In 2014, public repositories—which serve as "warehouses" at the heart of software supply chains—handled billions of download requests serving a global developer population of more than 11 million. [3]

Open source projects upload their open source components to a variety of warehouses where they can be freely shared by millions of developers anywhere, any time. These warehouses are commonly known in the software supply chain as public repositories such as the Central Repository, npmjs.org, rubygems.org, pypi.python.org, and NuGet.org. Here is a snapshot of recent use of public repositories:

Public Open Source Repositories	Download Requests Served (Annualized)
central.sonatype.org	17,213,084,947
npmjs.org	15,460,748,856
rubygems.org	4,959,638,830 (since inception)
NuGetGallery.org	280,124,916

Source: Each public repository provides statistics on activity levels for their communities.

The never-ending supplier ecosystem of components and versions

As mentioned earlier, the Central Repository is the source of record for Java and related open source software components. This repository is the most heavily used of its kind handling 17.2 billion requests in 2014 alone[4] and, as such, offers the widest-angle lens into the open source ecosystem and the software supply chains it serves. We have analyzed data from the Central Repository to help provide a better understanding of the software supply chain throughout this report.

Deeper analysis of the open source projects within the Central Repository showed more than 105,000 open source projects (categorized by Group-Artifact - GAs) are housed there[5]. When considering multiple versions of each component (categorized by Group-Artifact-Versions - GAVs), there were 834,399 components at the end of calendar year 2014[6]. As of June 2015, the number of component versions exceeded 974,000[7] and is well on its way to or above 1 million by the time you read this report. In the context of software supply chains, these projects represent the total number of suppliers while GAVs represent the total catalog of unique parts available.

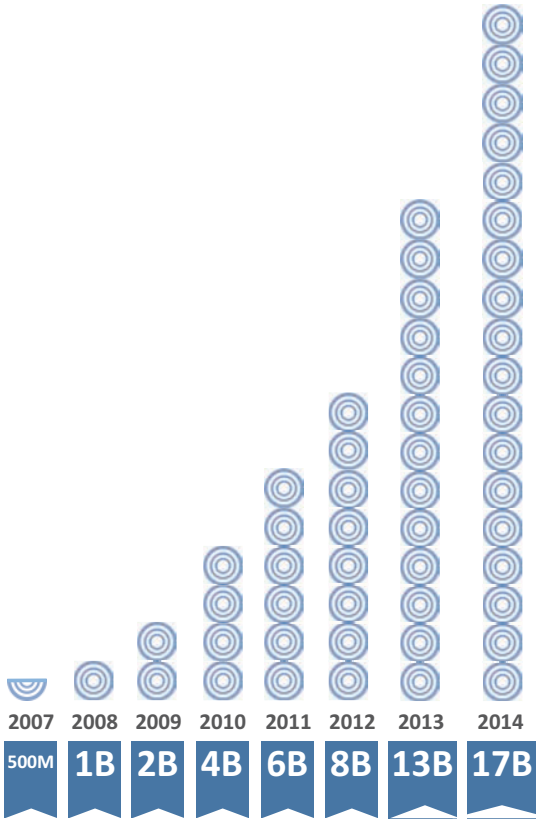


Image 4: The growth of download requests from the Central Repository have grown 30-fold since 2007. Source: Sonatype.

Analysis of download requests from the Central Repository reveals 106,087[8] organizations requested components in 2014. Automated requests can originate from a variety of sources, including popular build and design tools (Aether, Ant, Ivy, Leiningen,

Gradle, Maven, Eclipse), repository managers (Archiva, Artifactory, Nexus), and orchestration platforms (Puppet, Chef, Rundeck).

While the global base of open source projects has fed software supply chains for many years, they're not the only suppliers. As development organizations look to push the envelope on speed, they are looking for new sources for software, infrastructure, and services. For example, developers are looking for new ways to create Docker images and share them with other developers. For this purpose, Docker, Inc. runs Docker Hub, allowing companies to search and pull images as needed. Organizations like CoreOS Enterprise Registry provide a similar service.

As popularity of reusable containerized images, microservices, binaries, and code continues to grow, the supplier ecosystem will continue to expand to support this need.

Lack of meaningful communications channels

Unlike a traditional supply chain where the parts supplier and manufacturer have a clear relationship and communications channel, in a software supply chain that communication channel is not only broken, for many it simply doesn't exist.

While a handful of projects release new component versions weekly, on average a project will release new versions 3.5 times a year[9]. These releases might provide new functionality, improve performance, fix bugs, or occasionally patch newly found security vulnerabilities (which are generally slower to be remediated). The lack of a communications channel, coupled with the fact that a component is never taken out of circulation, makes the chances even greater that development teams will unknowingly use outdated, defective parts.

The lack of a communications channel makes the chances even greater that development teams will uknowingly use outdated, defective parts.

Choosing the Best Suppliers (Sourcing)

There is a robust and active ecosystem of suppliers providing a steady stream of innovative components to feed the demands of software supply chains. However, like in traditional manufacturing, not all suppliers deliver parts of comparable quality and integrity. Research shows that some open source projects use restrictive licenses and vulnerable sub-components, and some projects are far more diligent at updating the overall quality of their components.

Choosing an open source project supplier should be considered an important strategic decision, because changing a supplier is far more effort than swapping out a specific component version. Like traditional suppliers, open source projects have good and bad practices impacting the overall quality of their component parts. Where traditional manufacturing supply chains intentionally select specific parts from

approved suppliers, software supply chains rely on an unchecked variety of supply. Development teams choose whatever technology is deemed appropriate, as well as whatever version might be in vogue at the time of selection, introducing a significant degree of unnecessary complexity—and sometimes risk.

Hidden component defects

Not unlike traditional finished goods, a single part may rely on many other parts to function properly. In software development, components are like molecules, not atoms. A single component may rely on hundreds of other sub-components known as dependent components. A direct dependency (aka 1-hop) is a component directly tied to the core component. As shown in image 5, there are multiple layers of dependencies. This complex, inter-related web of components makes it even more difficult for some open source projects to manage vulnerabilities in their own software, even despite best efforts to produce quality code. As a result, neither the open source project—or the development teams who consume them—are aware of known vulnerabilities in dependent components.

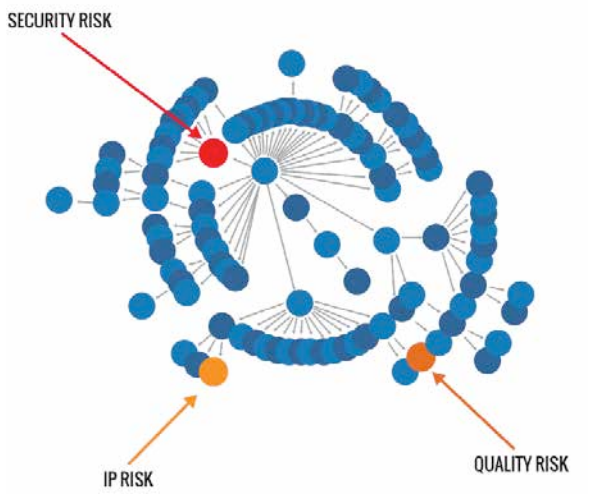


Image 5: A single component may rely on hundreds of other sub-components known as dependent components.

Mean-time-to-repair (M-T-T-R)

Overall, defect rates between open and closed source projects have been well documented and are believed to be comparable. But until 2014, little quantitative analysis had been available on the mean-time-to-repair (MTTR) defects within open source projects. In order to shed new quantitative light on this discussion, Sonatype initiated an analysis of open source project code bases hosted within the Central Repository.

Although the average open source project may release new component versions 3.5 times a year, it does not necessarily mean they are patching known security-related issues. In fact, in their seminal USENIX article *Almost Too Big To Fail*, Dan Geer and Josh Corman, shared findings from the Sonatype analysis. "An early analysis of open source projects with already identified vulnerable dependencies revealed some troubling behavior. Direct (aka '1-hop') vulnerable component dependencies were only remediated 41% of the time. Put differently, more than half (59%) of the vulnerable base components remain unrepaired. Folding multiple components into your projects means inheriting not just the components' functionality but also their (largely unrepaired) flaws. For the 41% that were fixed at all, the MTTR was 390 days (median 265 days). Filtering for just Common Vulnerability Scoring System (CVSS) level 10 vulnerabilities brought the mean of this subset down to 224 days. And this is just for 1-hop dependencies—there is as yet no mechanism to cause remediated flaws to flow automatically through the dependency graph, and there may never be." [10]

Hidden license complexity

Open source license risks are well known across the development community but are not always well understood. This is a key factor when choosing an open source supplier and component. According to long-time hosting provider SherWeb: "Taking what's already been created, adding to it, improving upon

it, and in turn sharing it with a worldwide community is what has allowed open source innovations to become 'arguably the single most influential body of software around the world.' But it has also left

Open source is everywhere, from mobile phones to medical devices to supercomputers to home appliances, and as such patent lawyers have never been busier.

many open to attacks from those who would claim the process by which they invented software is theirs and theirs alone. Open source is everywhere, from mobile phones to medical devices to supercomputers to home appliances, and as such patent lawyers have never been busier."[11]

Sonatype's analysis of the components in the Central Repository revealed that 34% included restrictive GPL licenses.[12] Image 6 developed by Ritambhara Agrawal of Intelligere provides a high-level description of the different licenses often assigned to open source and third-party software components.

Many lawsuits have arisen from the improper use of open source software, including well-known cases like BusyBox vs. Monsoon, Cisco vs. FSF, and Oracle vs. Google. While an improper license will not impact the performance of software, it can lead to enormous headaches, legal fees, and unplanned rework to remove the undesirable component.

Free Open Source Software (FOSS) Licenses

Liberal		Weak Copyleft		Non-Standard		Copyleft		
BSD 2-Clause License	Apache-2.0	Mozilla Public License 2.0	LGPL-2.1	JSON License	Sun Public License v1.0	GPL-3	Ruby License	AGPL-1.0
No restrictions on internal use.	No restrictions on internal use.	No restrictions on internal use.	No restrictions on internal use.	No restrictions on internal use.	No restrictions on internal use.	No restrictions on internal use so long as license remains in force.	No restrictions on internal use.	No restrictions on internal use so long as license remains in force.
Must retain copyright notice and include license.	Must retain copyright, patent, trademark, license, and attribution notices.	Must retain copyright notice and include license.	Must retain original copyright notice and include license.	Must retain copyright notice and include license.	Must retain copyright notice and include license.	Must retain original copyright notice and include license.	Must duplicate all of the original copyright notices and associated disclaimers.	Must retain original copyright notice and include license.
No obligation to disclose source code.	No obligation to disclose source code.	Must disclose all modified source code under the MPL license.	Must release all original and modified portions must with the source code to the downstream users.	No obligation to disclose source code.	Must make modified files available in source code.	Must release all original and modified portions with the source code to the downstream users.	Must disclose source code with distribution.	Must disclose source code when you publish, distribute, or serve modified software through a web portal.
	Must cause modified files to carry notice that You modified them.	Does not grant any rights in the trademarks, service marks, or logos of any Contributor.	Must cause the files modified to carry prominent notices stating that you changed the files.		Must cause all modified code to contain a file documenting changes made.	Must cause the files modified to carry prominent notices stating that you changed the files.		Must cause the files modified to carry prominent notices stating that you changed the files.
	If a NOTICE text file is included with the Work, then that NOTICE file must be included with distribution.			Non-Standard term: The Software shall be used for Good, not Evil.	Non-Standard term: All end user license agreements (excluding distributors and resellers) which have been validly granted by you or any distributor hereunder prior to termination shall survive termination.	Must accompany source code with the Installation Information.		

Source: Brian Fitzgerald, Sonatype, Inc.
Disclaimer: This table does not provide legal advice.

Image 6: There are a wide variety of open source licenses, each with a different set of obligations for the user.
Source: Brian Fitzgerald, Sonatype, Inc.

PARTS: OPEN SOURCE COMPONENTS AND LOCAL WAREHOUSES

Software components age more like milk than wine, so components that were once thought to be reliable can be discovered to be vulnerable in the future. Most troubling, organizations unwittingly continue to use known vulnerable components at an alarming rate. Analysis shows that 51,000 of the components in the Central Repository have known security or license concerns and that 6.2% of all components downloaded from Central last year included known vulnerabilities.[13]

Due to a lack of adequate visibility, tools and processes, most development organizations are unaware of the known vulnerabilities or license risks in the open source components downloaded from the public repositories. There is a generally accepted notion that "with many eyeballs, all bugs are shallow" in open source components. It is true that broad open source adoption means that many organizations are—in effect—testing the components, however as noted previously:

1. Once a component is shared in a public repository, it stays there forever even after many newer, safer versions have been introduced.

2. A once safe component may be found to be vulnerable at any time.

3. Components often depend on other components in order to function, much like an engine needs multiple other parts. If a vulnerability is found in a component dependency, it is generally very difficult for either the supplier (open source project) or the development teams to know about it—or track it down and fix it.

4. There is clear evidence that known vulnerable or defective components stored in public warehouses are downloaded by unknowing development teams, and end up in our software largely unnoticed.

How can known vulnerabilities be largely unnoticed? There is a fundamental lack of automation to put essential, relevant information in the right hands, at the right time, and at the right place.

PARTS
Open Source Components & Warehouses

DID YOU KNOW?

There are nearly 1 million unique components in the Central Repository alone.

51,000 components in the Central Repository have a known security vulnerability.

283,000 components in the Central Repository have known restrictive licenses.

Source: See Appendix, Figure 1.

	ORDERS	QUALITY CONTROL			
	Average Component Downloads (Orders)	Average Downloads with Known Vulnerabilities (Defects)	Percentage with Known Defective Parts	Component Downloads with Known Defects Older than 2013	Percentage with Known Defects Older than 2013
Average consumption by large financial or technology firms	240,757	15,337	7.52%	10,426	66.28%

Image 7: Defective component downloads by large financial services and technology firms in 2014. Refer to Figure 6 in the appendix for more details.

Developers simply don't have the time and are not incented to manually research each component. Other groups like open source review boards, legal and compliance teams, or security professionals are sometimes called in to support these efforts. But very few have the ability to track the current vulnerability or license status of components being used at the velocity in which they are being consumed.

The rapidly disseminated, global alert surrounding the OpenSSL Heartbleed vulnerability was an anomaly. While Heartbleed received international notoriety within a matter of days, there are an average of 50 security vulnerabilities found in open source components every day[14] that do not receive due attention. Authors Geer and Corman also discuss this topic in their USENIX article[15]: "The 'Legion of the Bouncy Castle Java Cryptography APIs' had a CVSS worst-case scenario fixed in April of 2008—more than six years ago. While CVE-2007-6721 is a severe security flaw in a security-sensitive project, nevertheless the unrepaired, vulnerable version was requested from Central Repository 42,124 times in 2014.[16]

While Heartbleed received international notoriety within a matter of days, there are an average of 50 security vulnerabilities announced in software every day that do not receive due attention.

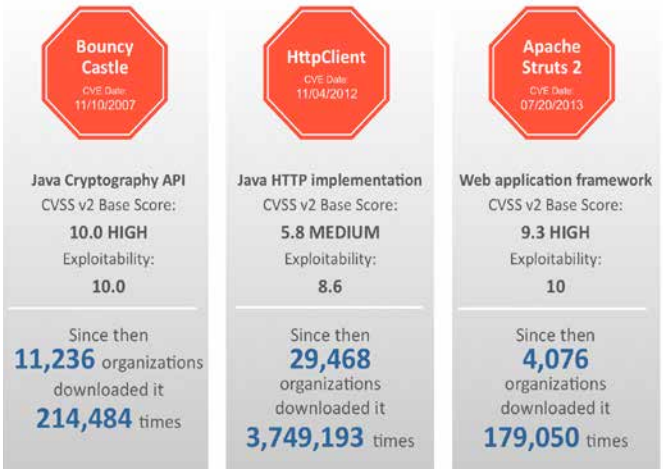


Image 8: Organizations continue to use known vulnerable components years after alerts have been issued. Source: Sonatype.

Geer and Corman continued, "Similar (disappointing) consumption patterns exist for Struts. Outside of CVE-2013-2251 compromised organizations, still vulnerable versions of Struts 2 continue to remain popular. Worse, Struts version 1-related artifacts still had 755,437 downloads in 2014,[17] despite its April 5, 2013 public warning. In other words, finding and fixing serious flaws in open source does not mean that the repaired versions are the ones that are used."

Placing these comments in the context of traditional supply chains, imagine the impact of sourcing parts with known defects. Can you imagine an automaker that sourced defective Takata airbags for use in a 2015 model? Or a laptop designer that sourced batteries that are known to catch fire? Or a pharmaceutical company that uses compounds that are known to cause birth defects?

Industry Spotlight:

Current Practices >>

One large company in the entertainment industry established a team to monitor and approve open source components being requested by thousands of in-house developers. However, an analysis of their download traffic from the Central Repository revealed that 97% of components sourced from public repositories were outside of the purview of their open source review board. This scenario is quite common. While component consumption is high and continues to grow, full visibility and control are extremely rare.

Best Practices >>

To help defend the U.S. government cyber infrastructure, and to help the Department of Homeland Security and other agencies carry out their cyber defense mandate, U.S. Congressional Representatives Ed Royce (R-CA) and Lynn Jenkins (R-KS) introduced the "Cyber Supply Chain Management and Transparency Act of 2014." The proposed cyber legislation aims to ensure that any organization selling software, firmware or products to the federal government is properly managing its software supply chain. The legislation recommended a focus on three supply chain practices:

1. Any software, hardware, or firmware sold to a procuring entity must provide a Bill of Materials of third party and open source components, including their versions.

2. Any software, hardware, or firmware cannot use known vulnerable components for which a less vulnerable component is available (without a written and compelling justification accepted by procuring entity).

3. Any software, hardware, or firmware must be patchable/updateable (within a reasonable timeframe)—as new vulnerabilities will inevitably be revealed.

Repository Management (Local Warehouses)

Repository managers are a fundamental first step toward software supply chain automation, focusing primarily on management of component "parts" within an organization. In this sense, they serve as local warehouses providing development teams with more efficient and controlled component access enterprise-wide. They are used to streamline the acquisition, consumption, sharing and deployment of components downloaded from public repositories as well as other internal build components and artifacts.

Some software development organizations are beginning to automate their supply chains, establish better best practices, and measure benchmarks. For example, there are more than 60,000 repository manager installations, such as Sonatype Nexus, Apache Archiva or JFrog Artifactory. These repositories act as local warehouses to host open source and proprietary parts to improve build performance and shorten supply chain cycles.

While many development organizations have adopted build tools like Maven, Ant, Ivy, and Gradle, most have yet to fully employ a repository manager, both to proxy remote repositories and to manage and distribute software components.

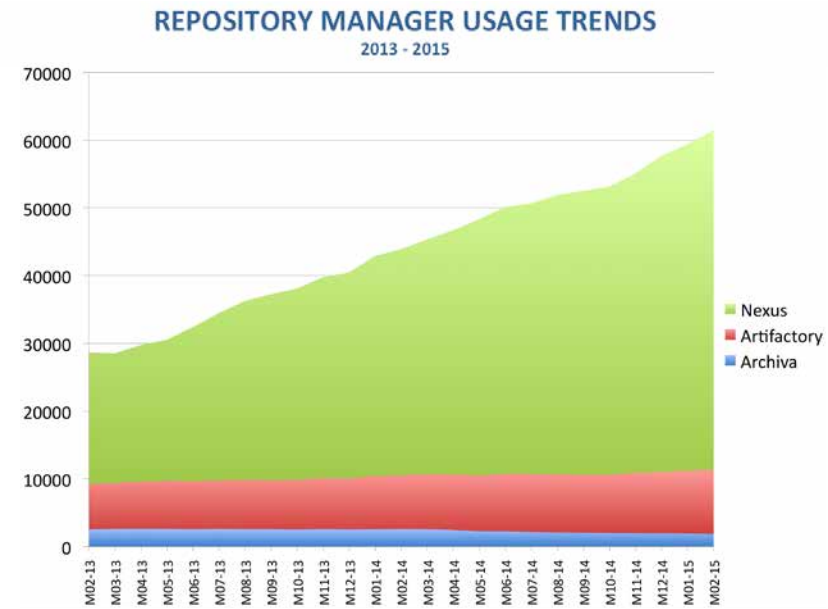


Image 11: Growth in repository manager installations 2013-2015.
Source: Sonatype's Central Repository

In a software development shop, any number of these tools or developers can be simultaneously downloading components at-will, which helps explain the practices that lead to multiple versions of the same component being downloaded.

In an ideal software supply chain configuration, build tools would interact with a repository manager to search for binary software components and retrieve software components on-demand. By caching and hosting components closer to the developers and tools that consume them, builds are faster, more efficient, and more reliable.

However, in most organizations build tools point to and retrieve components from public repositories directly (e.g., The Central Repository, NuGet Gallery, RubyGems.org). When analyzing the originating sources of downloads from the Central Repository, we see 95.24% of the requests coming from a variety of design, build, integration, and orchestration tools. The remaining

4.76% originate from repository managers.[18]

In a software development shop, any number of these tools or developers can be simultaneously downloading components at-will, which helps explain the practices that lead to multiple defects and versions being downloaded.

Shortening supply chains, speeding up development

If the software supply chain were fully optimized, we might expect to see the percentages in Image 13 flip. These percentages tell us that developers today are inefficiently sourcing virtually all of their components from distant central warehouses rather than targeting a source closer to home. Not only do long-distance requests take longer to fulfill, but they also add unnecessary bandwidth costs to operations.

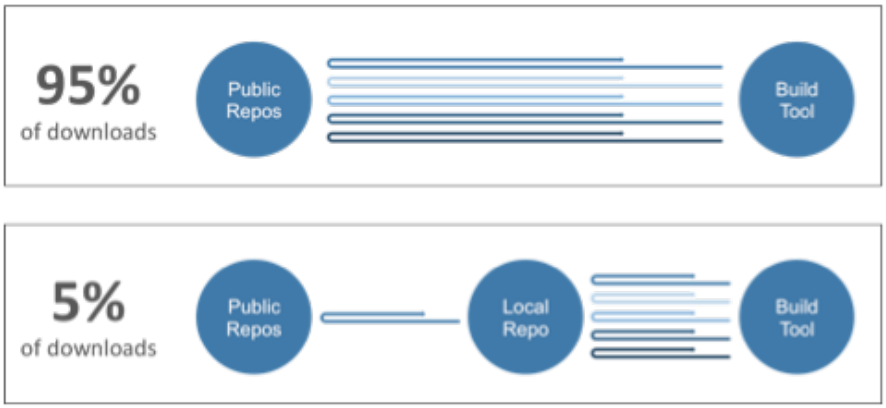


Image 13: Analysis of component distribution and sourcing reveals a significant level of inefficient behavior.

As a simple analogy, why would everyone in your neighborhood bypass the local grocery store and instead travel to a distant dairy farm in order to get their milk?

The impact of poor component consumption practices

While individual developers may not recognize the impact of their independent sourcing methods, the cumulative effect across hundreds or thousands of developers can add weeks or months to build times to development teams who are already pressured to deliver faster.

For example, downtime can become a big factor for development teams sourcing components directly from a public repository. While mature repositories like the Central Repository have high availability rates, some of the less mature component repositories experience minutes or hours of downtime each month. When these outages occur, all development relying directly on the repositories grinds to a halt. The frustration experienced by these developers is evident when social media channels like Twitter light up with developers in frustrated unison, calling "[XYZ] repository is down. Fix it now!"

Once cached in their repository manager, developers and their development tools can retrieve the component locally, as many times as necessary across the organization regardless of the number of applications needing it. Repository managers support the concept of download-once-use-many-times that improves sourcing practices and therefore are a foundational step in a broader trend towards improving performance and control of components across the software supply chain.

Why would everyone in your neighborhood bypass the local grocery store and instead travel to a distant dairy farm in order to get their milk?

The evolving role of repository managers in the software supply chain

To support continuous delivery, many organizations prefer to keep everything required to deploy or re-create an application's binaries—and the environments in which they run—in a well-known, stable, and easily accessible location. Where for many years, developers have stored binary artifacts in their repository managers, they are now expanding its role to include other artifacts created and used across the continuous delivery pipeline.

Repository managers are now used as the internal "parts warehouse" for tests, database scripts, build and deployment scripts, virtual machines, containers, documentation, libraries, configuration files for your application, and so on. The idea is that at all times a project has a system of record for all binaries, environments, and executable deliverables that are known to be safe for development and deployment.

Continuous Delivery Ltd.'s Dave Farley shared an example (see Image 13) of the artifact repository (a.k.a., repository manager) at the heart of a continuous deployment pipeline. In his example, the artifact repository (a.k.a., repository manager) acts as the central system of record for all inputs and outputs of the software supply chain's production line.

Repository managers are now used as a the internal "parts warehouse" for tests, database scripts, build and deployment scripts, virtual machines, containers, documentation, libraries, configuration files for your application, and so on.

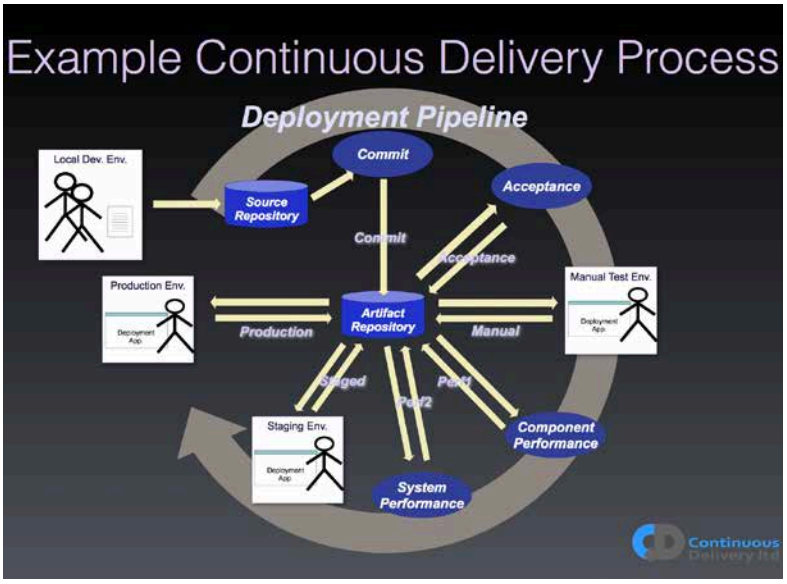


Image 13: Artifact repositories play a central role in deployment pipeline. Source: Dave Farley, Continuous Delivery Ltd. 2015. Slides at: http://bit.ly/1C9y7X8

Synchronizing teams across the software supply chain

Most anyone who has done maintenance programming has had the experience of not being able to recreate a defect because a change in one of the tools makes the original binary irreproducible. The discipline of using a repository manager also ensures that everyone is using the same set of documents and tools in development. This approach reduces the possibility that team members in other locations or overseas are using different requirements, building on new versions or a newer version of the compiler, etc.

In most organizations, more than one repository manager is used. The repository managers then are synchronized to ensure all development teams have access to the same artifacts for development and production. When implemented correctly, everyone on the team is essentially drawing from the same warehouse.

Using version control for all production artifacts

In leading DevOps and continuous delivery practices, it's easy to recreate environments for testing and troubleshooting. Absolutely all artifacts used to create production environments and the application that run in them are version controlled. The ability to get changes into production repeatedly in a reliable, low-risk way depends on the comprehensive use of version control.

Similar to practices in other highly-tuned supply chains, leading software development teams also pursue strict version controls and traceability for all of their components. The breadth of version consumption is limited and if any quality defects are later identified in components that were used, they can be found in minutes versus weeks.

Industry Spotlight:

Current Practices >>

Analysis of some of the world's largest financial services and technology firms demonstrates that use of repository managers alone does not improve the quality of components made available to developers. While companies downloaded an average of 29,697 components to their repository managers, on average, in these large organizations 7% of all component requests had known defects.[19] While many companies pursue "Golden Repository" strategies, where attempts are made to keep only approved components in their repository managers, we now have quantifiable evidence to show the approach does not work without additional rigor.

Best Practices >>

A global investment banking titan had empowered their developers to consume new components at-will, through any available channel. When they learned that thousands of in-house developers bypassed their repository manager to download over 2.6 million components, they recognized the inefficiency of their sourcing practice as well as the potential for voluntarily inheriting quality, license, and security issues. The bank moved quickly. They required developers to first query local repository managers for components and, within one year, the bank reduced Central Repository downloads from 2.6 million to 95,000 components. By reducing component downloads, the company eliminated complexity and inefficiency from their software supply chain which lead to an estimated yearly savings of 30 days of build time.

MANUFACTURERS: ASSEMBLED SOFTWARE DEVELOPMENT

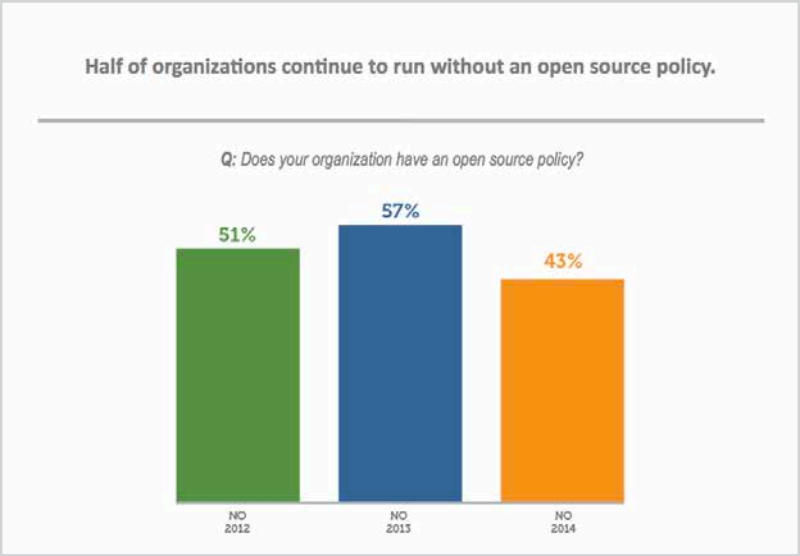
Most development teams strive for ever-increasing speed and throughput. Yet the software assembly process remains rife with inefficiencies, largely due to a lack of enforceable policies and guardrails to help developers make better, safer decisions. This is not a "people problem." This is an automation problem.

Very few CIOs, software development executives, enterprise architects, and especially personnel residing outside of the IT organization realize the extent of their organization's reliance on the components and the supply chain that serves them. While legal, security, audit, open source review boards, and other functional organizations have attempted to detail and track consumption behaviors, they often fall short of gaining full visibility.

Time pressures can trump quality

Developers take great pride in their innovations, however time pressures often trump quality especially when time-consuming manual research is required to properly analyze component suitability. According to the results of of a 2014 Open Source and Application Security survey[20] of over 3,300 developers and IT staff.

- 43% of organizations don't have development policies that address open source and third-party component usage across versions, age, licenses, and security issues.



Source: Sonatype 2014 survey[20]

MANUFACTURERS Software Development Teams

DID YOU KNOW?

1 in every 16 component downloads included a known security vulnerability.[4]

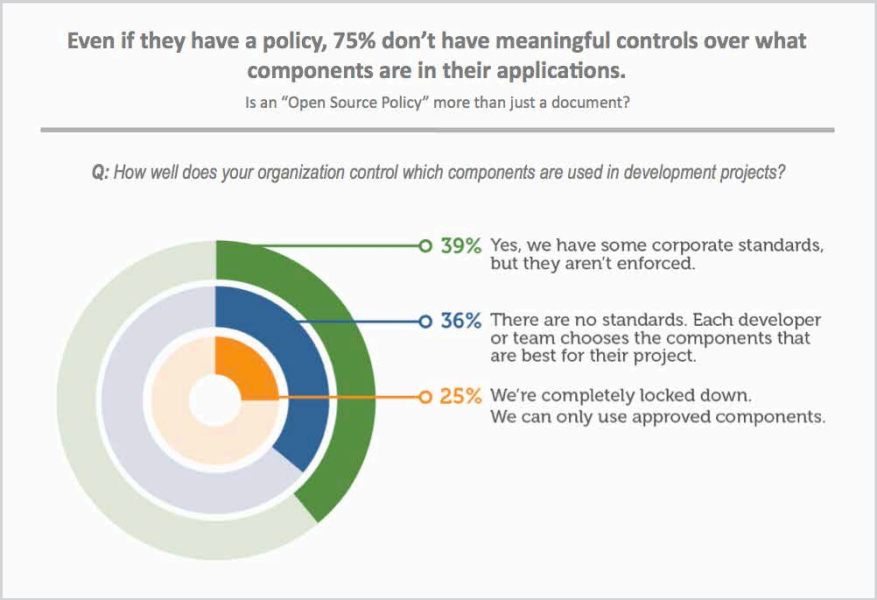
43% don't have policies to manage component quality.[20]

75% of those with policies don't enforce them.[20]

31% have had or suspect a breach in an open source component.[20]

Source: 2014 Sonatype Open Source Development and Application Security Survey.

- Of the organizations with policies in place, 75% of participants indicated that the policies were not enforced.



Source: Sonatype 2014 survey[20]

- In organizations with more than 500 developers, the situation was somewhat better as 73% of these organizations had a policy in place and 49% stated that use of open source and third-party components was strictly enforced.[21]

Automating acceptability

Organizations wishing to improve the quality and integrity of components being consumed must rely on automation. In light of the volume and velocity of component consumption we have already noted, clutching to current manual processes cannot enable an organization to reach new levels of productivity and competitiveness.

DevOps and Continuous Delivery teams should not just go through a checklist of acceptability, but also actually define the attributes that make a software component acceptable for use. Automated policies can then enforce things like "don't allow anything into my billing process if it has a severe security defect," or "don't allow anything into my customer-facing applications that leverage the fair use GPL licenses." Humans need only define the policies, then use automation for enforcement. Then humans can manage the occasional exception to the automated process.

Very few CIOs, software development executives, enterprise architects, and especially personnel residing outside of the IT organization realize the extent of their organization's reliance on the components and the supply chain that serves them.

Technical Debt: Assembly Line Inefficiencies

In one large financial services firm, developers had downloaded 81 of the 85 versions of the Spring Core framework in 2014. This not only means the investment bank had access to a wide variety of outdated versions, but that their developers potentially and unknowingly added unnecessary technical bloat and maintainability challenges to their portfolio.

Within the top 100 most popular components, we saw that 27 versions of a single component were downloaded by an average organization in 2014 alone.[22] By consuming the poor quality, risky, or defective components across their supply chains, organizations were electively building in technical and security debt. And since few organizations keep track of which components were used in each application, the time, effort, and cost of identifying and remediating those risks is substantial.

Imagine if a company manufacturing insulin pumps was able to freely select any one of 27 different pumping mechanisms. The impact is far reaching: it impacts the assembly line, customer service, and quality control. The same is true for software.

The cost of context switching

As development teams move toward continuous delivery and agile practices, they are even more likely to work on multiple projects in a given year. When you

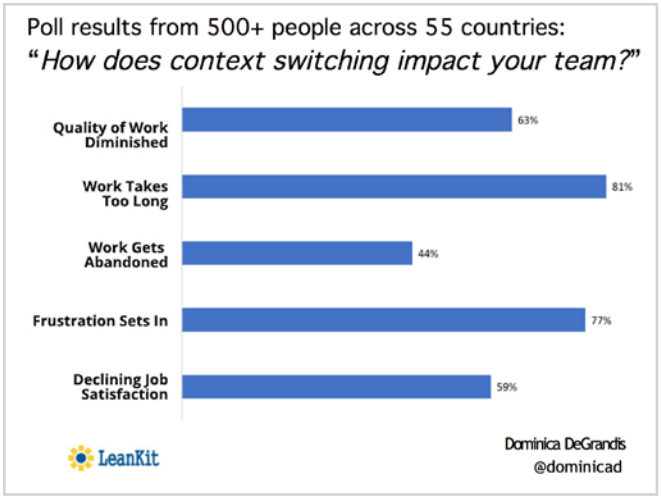


Image 10: The impact of context switching on development teams. Source: Dominica deGrandis, DevOps Days Austin 2015

are not only switching from one project to the next, but also switching between multiple component versions, the impact of context switching is even greater. As image 10 shows, context switching is proven to not only impact development speed, but also quality and job satisfaction.

Industry Spotlight:

Current Practices >>

While assisting a federal government organization in analyzing components in a large web application that helps service millions of citizens, an auditor discovered 11 different logging framework components in use. This behavior is similar to an automobile manufacturer using four different versions of door locks from four unique suppliers in one vehicle.[23]

Best Practices >>

A global Internet search and advertising giant uses tens of thousands of open source and third-party components across its developer community. To limit the complexity of its operations, developers are free to use any high-quality software component, but the company restricts them to using only one of the latest, most stable versions. By limiting the number of versions used, developers can easily shift between projects without needing to learn the nuances of an alternative version. It also takes the company less time to identify and repair defects.[24]

FINISHED GOODS: SOFTWARE APPLICATIONS

In the traditional manufacturing supply chain, product quality and safety are enhanced through diligent tracking of the suppliers and parts used in each product. However, in the software supply chain, where an average of 106 components[25] comprise 80-90% of the total application, few organizations have visibility into what components were used and where.

The Volume of Elective Re-work and Risk

Known defective components lead to quality and integrity issues within applications. While developers save tremendous amounts of time by electively sourcing software components from outside their organizations, they often don't have time to check those component versions against known vulnerability databases, internal open source policies, and other sources of quality information. Earlier in this report, we showed 6.2% of downloads from the Central Repository—at the front of the software supply chain—were components that included known vulnerabilities.

Components	Known Critical or Severe Security Vulnerabilities	Known restrictive licenses
106	24	9

Image 9: Components and known security or license risks in an average application

Analysis of over 1,500 applications reveals that by the time the applications are developed and released at the end of the software supply chain, a typical application has 24 known severe or critical security vulnerabilities and 9 restrictive licenses.[26]

Likewise, in a security analysis across 5,300 applications, Veracode also found and confirmed that an average application has 24 known security vulnerabilities associated with open source and third-party components.[27]

Where most organizations have relied on manual reviews of open source components used in applications, these practices are proving insufficient. Based on the volume and velocity of open source and third-party software components being consumed, it is impossible to check everything manually. It is simply too expensive and too slow—especially given the sub-components or dependencies which are less obvious.

FINISHED GOODS
Software Applications

DID YOU KNOW?

23% of the components in the average application have critical or severe known vulnerabilities.[26]

There are 9 restrictive licenses per application, critical or severe.[26]

63% of organizations keep an incomplete software Bill of Materials.[28]

Software Bill of Materials

Unlike "bill of materials software" which is used in traditional manufacturing supply chains to list the suppliers and parts used in a product, a "software bill of materials" (BOM) is an inventory of the third party and open source components used to build an application.

As noted in Wikipedia, "The concept of a BOM is well-established in traditional manufacturing as part of supply chain management.[29] A manufacturer uses a BOM to track the parts it uses to create a product. If defects are later found in a specific part, the BOM makes it easy to locate affected products.

"A software BOM is useful both to the development organization (manufacturer) and the buyer (customer) of a software product. Builders often leverage available open source and third-party software components to create a product; a software BOM allows the builder to make sure those components are up to date and to respond quickly to new vulnerabilities.[3] Buyers can use a software BOM to perform vulnerability or license analysis, both of which can be used to evaluate risk in a product. Understanding the supply chain of software, obtaining a software BOM, and using it to analyze known vulnerabilities are crucial in managing risk." [30,31]

A software bill of materials not only inventories what is used, but in some cases it also syncs with real-time component defect data to indicate which components have known vulnerabilities or license risks. Various software supply chain automation tools expand the bill of software much further, alerting stakeholders automatically when a defect alert occurs. In advanced tools, the entire software lifecycle is automated to ensure that defective components are avoided and continuous monitoring instantly identifies newly announced vulnerabilities as soon as they are discovered.

Quality Controls: OWASP, PCI, FS-ISAC, U.S. Congress

An executive from a well-known U.S. federal agency once said: "There is no building code for building code." While that is not 100% true, it does accurately describe the component-based portion of modern software development. Among the many benefits of supply chain automation in other industries, quality controls have made products safer and more reliable. However, in the software supply chain, the lack of controls can, in some cases, result in critical security defects that may put consumers at risk of losing their credit card data or healthcare history to the highest bidder in the black market. In the case of defective open source components in medical devices or cars, the risks can be even greater.

Recognizing the growing and ongoing threat posed by the use of defective component parts, organizations representing IT, financial services, healthcare, and application development have proposed new guidelines for ensuring the highest-quality parts are selected, used, and traceable across application portfolios. These include:

The Open Web Application Security Project (OWASP): In 2013, OWASP updated its top ten list of application security threats to include A9, which advises against "using components with known vulnerabilities." The full description from OWASP states: "Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server

takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts."[32]

Payment Card Industry (PCI): PCI standards help ensure that banks, financial services firms, and merchants protect their customers' credit card data. PCI guidelines were quickly updated following the OWASP A9 guidelines addressing known vulnerabilities.

Financial Services Information Sharing and Analysis Center (FS-ISAC): The FS-ISAC started the Product & Services Committee to identify appropriate security control types for third-party service and product providers. This effort is due to the fact that the application represents the "new perimeter." The working group references Gartner research that states, "Since enterprises are getting better at defending perimeters, attackers are targeting IT supply chains." The FS-ISAC report also stated, "Recent breach reports such as Verizon's Data Breach Investigations Report underscore the vulnerability of the application layer, including third-party software. This new perimeter of third party software must be addressed."[33]

The FS-ISAC committee addressed three suggested control types that should be implemented based on the new supply chain reality: vBSIMM process maturity assessment, binary static analysis, and policy management and enforcement for consumption of open source libraries and components.

U.S. Congress: To help defend the U.S. government cyber infrastructure and help the Department of Homeland Security and other agencies carry out their cyber defense mandate, U.S. Congressional Representatives Ed Royce (R-CA) and Lynn Jenkins (R-KS) introduced the "Cyber Supply Chain Management and Transparency Act of 2014." The proposed cyber legislation aims to ensure that any organization selling software, firmware, or products to the federal government is properly managing its software supply chain. The legislation recommended a focus on three supply chain practices: (1) any software, hardware, or firmware sold to a procuring entity must provide a bill of materials of third-party and open source components, including their versions; (2) any software, hardware, or firmware cannot use known vulnerable components for which a less vulnerable component is available (without a written and compelling justification accepted by procuring entity); and (3) any software, hardware, or firmware must be patchable/updateable (within a reasonable timeframe)—as new vulnerabilities will inevitably be revealed.[34]

LESSONS LEARNED FROM TRADITIONAL MANUFACTURING SUPPLY CHAINS

What if manufacturers built cars using the same processes currently used to build software? They could choose any part from any supplier they wanted. They could choose these parts without visibility into quality, or which parts or versions their colleagues were using. Every car would likely have a unique set of parts and a wide variety of known defects already built in as it rolled off the assembly line. Months or years later, when a defect was widely publicized, they wouldn't know if they used that part and, if so, where.

Software is not the first industry to face supply chain challenges. From automotive to energy, healthcare to defense, mature industries have automated their supply chain to produce goods quickly and efficiently with high quality and minimized risk. The benefits include improved performance, profitability, transparency, and sustainable competitive advantages.

Leading organizations like Toyota learned they could sustain a competitive advantage by following three basic principles—use fewer and better suppliers, use higher quality parts, and track what is used and where.

Toyota has reduced complexity by using only 125 plant suppliers for the Prius. General Motors has 800 for the Volt. General Motors produces 54% of the content of their vehicles while Toyota produces 27%.

General Motors has 20x the suppliers and yet they produce half of the content of their vehicles. Therefore, it's no surprise that the Volt cost nearly $35,000 and a Prius less than $25,000. At the time of this research, Toyota sold 20,000 Prius units a month and General Motors sold 1,700 Chevy Volts.[35]

Interestingly, the software development practices of the majority of organizations today tend to have more in common with General Motors than you might think. And in fact, it is far more troubling because—in a typical software supply chain—developers don't have visibility into component quality, yet each developer can choose any component they want. Freedom of choice combined with lack of quality information is a toxic combination. (See appendix, figure 5 for more information.)

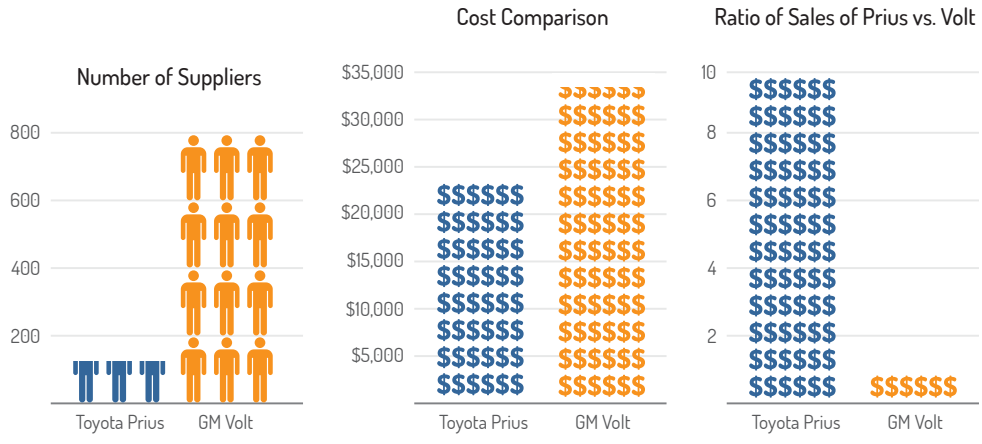


Image 2: Supplier complexity impacts cost and quality, two important factors for competitive differentiation. The same is true of software. Source: Toyota Supply Chain Management: A Strategic Approach to Toyota's Renowned System, by Ananth Iyer and Sridhar Seshadri

AUTOMATION: HOW TO IMPROVE SOFTWARE SUPPLY CHAINS

Unnecessary—and largely hidden—complexity permeates nearly all software development today. If it continues along the same path, these speed bumps will block further increases in productivity and the on-time, on-budget delivery of quality software.

The introduction of high-level languages, object-oriented programming, agile, continuous delivery, DevOps, and dependency management are all ways in which software development practices have responded to problems with scalability, complexity, and the desire to unlock the next level of efficiency.

Software supply chain complexities, coupled with the volume and velocity of open source and third-party components flowing through them, have created such a requirement. Soon, the industry at large will begin to demand solutions in response. But the answer is here today: automation.

Mother Teresa said, "If I look at the mass, I will never act. If I look at the one, I will." Looking at the massive volume and velocity of open source and other artifact consumption can be discouraging and overwhelming. However, the volume and velocity within our software supply chains will not diminish—and without a new approach, the volume of unchecked quality and integrity of parts being consumed will continue to build up as technical debt.

Automation in areas of testing, build, and deployment has provided significant performance benefits. Likewise, investments in software supply chain automation have shown markedly improved efficiency and controlled risk, as the best practices in this report illustrate. Automation can unleash the potential of an organization's development capacity. Rarely is there such an opportunity to simultaneously increase speed, efficiency, and quality.

Solutions that facilitate comprehensive software supply chain automation are poised to usher in the next wave in development productivity—with gains on par or even greater than possible with agile, Lean, and DevOps.

To begin improving your software supply chain, consider these next steps:

1. **Create a software bill of materials for one application:** Visibility into one application can help you better understand your current component usage. A number of free and paid services are available to help you create a software bill of materials within a few minutes. The bill of materials will help you to identify the unique component parts used within your application and the suppliers who contributed them. These reports list all components used, and several services also identify component age, popularity, version numbers, licenses, and known vulnerabilities.

2. **Take inventory within one of your local warehouses:** Repository managers (a.k.a., artifact repositories) are commonly used by software development teams. Components residing in repository managers can easily be reused across software development. If outdated or defective components reside in the repository manager, they can easily make their way into multiple applications. Start by identifying a repository manager in use within your organization and creating an inventory of the parts cached there. Some repository managers

automate this inventory reporting process for you, listing components, versions, licenses, and security vulnerabilities.

3. **Design approval processes to be frictionless, scalable, and automated:** Manual reviews of components and suppliers cannot keep pace with the current volume and velocity of consumption. Your organization must not only define your policies for supplier and part selection but also find practical ways to enforce them without slowing down development or inadvertently encouraging workarounds. Policies must be agile enough to keep pace with modern development. Strive to automate policy enforcement and minimize drag on developers.

4. **Enable developer decision support:** Developers are primary consumers of components in software supply chains. They initiate every component request. Help developers by automating the availability of information on component versions, age, popularity, licenses, and known vulnerabilities within their existing development tools so it is easy to pick the best components from the start. By selecting the highest-quality components from the highest-quality suppliers early, you will improve developer productivity and reduce costs.

5. **Ensure visibility and traceability throughout the software lifecycle:** Components can enter the software development lifecycle at many different places. Requests are initiated by developers and the tools they use regularly to build, integrate, and release applications. Since component selection is not a point-in-time event, continuous monitoring should be used to alert you when new and unchecked components have entered the supply chain. Alerts can also tell you when components used are out-of-date or when new vulnerabilities have been discovered. Continuous monitoring will also allow for component traceability, improving overall mean-time-to-repair defects.

Organizations that have taken control of their software supply chains have seen tremendous gains in developer productivity, improved quality, and lower risk. This report has highlighted a number of industry-leading practices. As your organization embarks on its journey to supply chain automation, you can use these examples to establish new performance, quality, and productivity benchmarks for your organization.

ABOUT SONATYPE

Sonatype helps organizations build better software, even faster. Like a traditional supply chain, software applications are built by assembling open source and third party components streaming in from a wide variety of public and internal sources. While re-use is far faster than custom code, the flow of components into and through an organization remains complex and inefficient. Sonatype's Nexus platform applies proven supply chain principles to increase speed, efficiency and quality by optimizing the component supply chain. Sonatype has been on the forefront of creating tools to to improve developer efficiency and quality since the inception of the Central Repository and Apache Maven in 2001, and the company continues to serve as the steward of the Central Repository serving 17.2 Billion component download requests in 2014 alone. Sonatype is privately held with investments from New Enterprise Associates (NEA), Accel Partners, Bay Partners, Hummer Winblad Venture Partners and Morgenthaler Ventures. Visit: www.sonatype.com

APPENDIX

Figure 1: The Volume and Size of the Global Software Supply Chain based on data compiled from the Central Repository.

This chart provides visibility to the number of suppliers, parts, and warehouses used across the software supply chain, specific to Java open source components. The volume of download requests, including those with known vulnerabilities, has been noted in the chart. The chart represents data analyzed from the Central Repository (www.sonatype.org) from calendar year 2014.

Supply Chain	Volumes Measured in 2014	Software Supply Chain Translation
Parts Consumed	17,200,000,000	Download requests of open source components managed in 2014
Suppliers	>105,000	The number of open source projects supplying new components as measured by Group-Artifact, as of year-end 2014
Total Parts Available	>834,000	Total number of parts (Group-Artifact-Version) when considering all versions, as of year-end 2014
Central Warehouses	>100	Estimated number of large, public open source repositories
Local Warehouses	>60,000	Repository managers caching and hosting open source and proprietary components, as of February 2015
Manufacturers	>106,000	In 2014, more than 106,000 software development organizations downloaded open source components from the Central Repository
Average number of unique part versions ordered (for top 100 parts by order volume)	27	Sonatype assessed the top 100 most popular component downloads across 29 large companies in the financial services and technology industry. This is the average number of versions of open source component parts consumed in 2014 (unique Java/Maven parts are identified by Group-Artifact-Version)
Number of parts with known defects residing in the Central Repository	>51,000	Open source components with known security vulnerabilities housed with the Central Repository. 51,000 (6.1%) components with known vulnerabilities resided in the Central Repository, as of year-end 2014
Percent of parts with known defects delivered in 2014	6.22%	Total percentage of downloads from the Central Repository that included components with known security vulnerabilities. Roughly 1 in 16 downloads included a known vulnerability
Percent of components with known restrictive licenses in the central warehouse	34.02%	Total percentage of components residing in the Central Repository that include a type of GPL open source license. 283,800 components had a GPL type of license, as of year-end 2014

Figure 2: Target Benchmarks for Software Supply Chain Best Practices - Quality Control

Sonatype research and analysis has revealed notable differences in quality control practices and policies across organizations. While the best practice would be to eliminate all use of known vulnerable or poor-quality components, Sonatype recognizes that as an end goal. Industry guidelines developed by OWASP, FS-ISAC, and PCI recommend not using components with known vulnerabilities in applications. While a number of organizations have made significant progress on this front, Sonatype decided to highlight some of the better examples of policies and practices, understanding that we have significant room to improve across all industries.

Supply Chain Behaviors	Observations of Better Software Supply Chain Behaviors	Current Software Supply Chain Behaviors
Quality Control - versioning	The best mandate using no more than 1 or 2 versions per component	A global Internet search and advertising company is known to mandate the use of one version of each component used in their software development practice. *Source: Presentations by Gene Kim and Josh Corman at RSA Conference, April 2015.* By comparison, some companies analyzed had consumed an average of 27 versions of a single component in 2014. These 27 versions were noted among the top 100 components downloaded by those companies in 2014. *Source: Analysis of data from the Central Repository, April 2014 - March 2015*
Quality Control - policies	47% of large companies strictly enforce quality through policies	In firms with 500 or more developers, 73% had a policies in place to support selection of high quality (functionality, version, age, security, license) software components. Within these same firms, 47% of firms strictly enforced that policy. By comparison, 57% of all companies have a policy in place, but 75% claim to not strictly enforce it. *Source: 2014 Open Source Development and Application Security Survey*
Quality Control - track and trace	67% of companies with policies in place track and trace all components in use	67% of companies with an open source policy in place track and trace all components and their dependencies used in an application. In companies with 500+ developers, the percentage dropped to 49%. *Source: 2014 Open Source Development and Application Security Survey* By comparison, 60% of organizations have loose or no controls over component versions, including dependencies, used in their environments. That is, they claim to have no software bill of materials or an incomplete software bill of materials. *Source: 2014 Open Source Development and Application Security Survey*
Quality Control - known defects	3.25% of components downloaded / sourced included defects	One company sourcing more than 400,000 components from the Central Repository in 2014, only sourced 3.25% with known security defects (elective risk). By comparison, cross-industry rates reached 6.22% of overall downloads. One company that sourced over 24,000 parts reached a known defect volume about 14%. It may only take one CVE to have a breach or one misused GPL to have a legal settlement. *Sources: Analysis of 2014 downloads from the Central Repository. Analysis included evaluation of the top 500 organizations downloading components in 2014, as measure by volume. Sonatype also analyzed 2014 download data from 29 large financial services and technology companies.*
Quality Control - known defects over time	When downloading components with known CVEs, 52% included vulnerabilities dated 2013 or older.	Companies are electively sourcing components with known security defects. When analyzing the quality and age of these defects, the best performer in our analysis of 29 large financial and technology firms demonstrated 52% of vulnerable downloaded components had CVEs dated 2013 or older. *Source: Analysis of 2014 download data from 29 large financial services and technology companies* By comparison, the cross-industry average of CVE downloads dated 2013 or older was 77%. Components with CVEs dated 2010 or older accounted for 17% of known vulnerability downloads. *Source: Analysis of downloads from the Central Repository in 2014.*

Figure 3: Target Benchmarks for Software Supply Chain Practices - Efficient Distribution

Sonatype research and analysis has revealed notable differences in the sourcing of open source components by development teams. These sourcing practices sometimes represent efficient distribution of components across the software supply chain by utilizing local warehouses (i.e., repository managers). Inefficient sourcing practices would bypass the local repository manager where the requested component was stored and go directly to the distant public repository (i.e., central warehouse) for retrieval of component parts.

Supply Chain Behaviors	Observations of Better Software Supply Chain Behaviors	Current Software Supply Chain Behaviors
Efficient Distribution - via local warehouse	The best achieve 99% of components sourced from local repositories	Of the top 500 organizations (by volume) downloading components from the Central Repository in 2014, the best source 99% of their components through a repository manager to ensure faster, more reliable builds. By the time we reach the 30th ranked downloader to repository managers (by volume), the figure drops to 25% of components sourced from repository managers. By comparison, the cross-industry average is 4.76% of components sourced from repository managers. *Source: Analysis of 2014 downloads from the Central Repository. Analysis included evaluation of the top 500 organizations downloading components in 2014 as measure by volume.*

Figure 4: Analysis of Components Used within Applications

This chart provides a summary of more than 1,500 applications analyzed by software developers using Sonatype's Application Health Check (AHC), a community service offering available at no cost. The AHC details component popularity, age, license types, and known security vulnerabilities as part of providing a software bill of materials.

Supply Chain	Volumes Measured in 2014	Software Supply Chain Translation
Average number of parts assembled into a finished product	106	Average number of open source components identified in an application
Number of known critical or severe security vulnerabilities in a typical application	24	Average number of open source components identified in an application that include known security vulnerabilities. (Note: if considering Java Runtime Environments or Operating System level defects, the average might be much higher)
Number of known restrictive license in a typical application	9	Average number of open source components identified with restrictive GPL license types, per application

Figure 5: Multiple Versions of Parts Often Downloaded by the Largest Development Teams

Sonatype analyzed the 2014 component downloads of 29 large companies in the financial services and technology industry. Those companies downloaded different volumes ranging from 10,000 to 900,000 components. Sonatype grouped the organizations in volume download bands to compare the number of orders, suppliers, and the variety of parts used across their software supply chains.

Orders		Suppliers	Parts	Variety
April '14 - April '15 Software Component Downloads[19]	Orders (Downloads)	Suppliers (Artifact)	n version of Parts (Version)	Average versions across top 100 component downloads
Range: 500K - 900K	622,517	7,927	27,698	35
Range: 150K - 499K	259,995	16,602	30,493	34
Range: 40K - 149K	58,350	3,951	11,457	23
Range: 10K - 39K	22,165	1,923	4,810	16
Average	240,757	7,601	18,614	27

Figure 6: Volume of Defective Parts Used

Sonatype analyzed the 2014 component downloads of 29 large companies in the financial services and technology industry. Those companies downloaded different volumes ranging from 10,000 to 900,000 components. We grouped the organizations in volume download bands to compare the number defects parts (i.e., components with known vulnerabilities) downloaded from the Central Repository. We also analyzed the age of CVE downloads.

	Orders	Quality Control			
Download (Range)	**Average Component Downloads (Orders)**	**Average Downloads with Known Vulnerabilities (Defects)**	**Percentage of Known Defective Parts**	**Component Downloads with Known Defects Older than 2013**	**Percentage of CVE Downloads with Defects Older than 2013**
Range: 500K - 900K	622,517	39,622	6.42%	27,621	69.92%
Range: 150K - 499K	259,995	14,514	5.83%	9,634	66.75%
Range: 40K - 149K	58,350	5,328	9,43%	3,184	59.01%
Range: 10K - 39K	22,165	1,885	8.41%	1,265	69.44%
Average	240,757	15,337	7.52%	10,426	66.28%

Figure 7: Comparison of Impact of Supply Chain Complexity on Prius versus Volt

Other industries have reduced supply chain complexity by using the highest quality parts from fewer and better quality suppliers. One example highlighted in the book "Toyota Supply Chain Management" by Ananth Iyer and Sridhar Seshadri reveals the advantages Toyota's Prius recognized over GM's Chevy Volt.

	Advantage	Toyota Prius	Chevy Volt
Unit Cost	61%	$24,200	$34,345
Units Sold	13x	23,294	1,788
In-House Production	50%	27%	54%
Plant Suppliers	16% (10x per)	125	800
Firm-Wide Suppliers	*4%*	*224*	*5,500*

Figure 8: Efficient Sourcing Practices By Manufacturers

Sonatype analyzed the 2014 component downloads of 29 large companies in the financial services and technology industry. Those companies downloaded different volumes ranging from 10,000 to 900,000 components. We grouped the organizations in volume download bands to compare the volume of components being downloaded by repository managers. Component downloads by a repository manager represent a more efficient practice of sourcing by software development organizations. Sonatype also analyzed the number of defective components being downloaded to these repository managers.

Download (range)	Volume of downloads by repository managers	% of total downloads requested by repository managers	Volume of downloads with known defects (CVEs) by repository managers	% of total downloads with known defects (CVEs) requested by repository managers
Range: 500K - 900K	63,353	11.25%	4,556	6.52%
Range: 150K - 499K	33,977	14.28%	2,213	6.17%
Range: 40K - 149K	19,061	28.31%	1,644	10.26%
Range: 10K - 39K	2,397	12.17%	192	6.19%
Averages	29,697	16.50%	2,151	7.28%

Footnotes:

1. Sonatype research including Application Health Checks and Open Source surveys, 2013 – 2014.
2. Openhub, https://www.openhub.net
3. IDC, http://www.idc.com/getdoc.jsp?containerId=prUS24529613
4. Sonatype analysis of the Central Repository for 2014
5. Sonatype analysis of the Central Repository for 2014. Statistics are updated daily at https://search.maven.org/#stats
6. Sonatype analysis of the Central Repository for 2014
7. Sonatype, https://search.maven.org/#stats
8. Sonatype analysis of the Central Repository for 2014
9. Sonatype analysis of the Central Repository for 2014
10. Almost Too Big to Fail, https://www.usenix.org/system/files/login/articles/15_geer_0.pdf
11. SherWeb, http://www.sherweb.com/blog/5-pivotal-open-source-lawsuits/
12. Sonatype analysis of the Central Repository for 2014
13. Sonatype analysis of the Central Repository for 2014
14. National Vulnerability Database, https://nvd.nist.gov
15. Almost Too Big to Fail, https://www.usenix.org/system/files/login/articles/15_geer_0.pdf
16. Sonatype analysis of the Central Repository for 2014. The original article provided stats for calendar 2013. The stats have been updated in this report to reflect calendar 2014.
17. Sonatype analysis of the Central Repository for 2014. The original article provided stats for calendar 2013. The stats have been updated in this report to reflect calendar 2014.
18. Sonatype analysis of the Central Repository for 2014
19. Sonatype analysis of 29 large financial services and technology sector companies and their Central Repository activity, April 2014 – March 2015.
20. Sonatype, Open Source Development and Application Security Survey 2014, bit.ly/2014OSS_survey
21. Sonatype analysis of Open Source Development and Application Security Survey 2014 results.
22. Sonatype analysis of 29 large financial services and technology sector companies and their Central Repository activity, April 2014 – March 2015.
23. Rugged DevOps: Going Even Faster with Software Supply Chains, presentation by Josh Corman and Gene Kim, RSA Conference, April 2015.
24. Rugged DevOps: Going Even Faster with Software Supply Chains, presentation by Josh Corman and Gene Kim, RSA Conference, April 2015.
25. Sonatype, 2014 analysis of Application Health Check results.
26. Sonatype, 2014 analysis of Application Health Check results.
27. Veracode, http://www.veracode.com/open-source-and-third-party-components-embed-24-known-vulnerabilities-every-web-application-average
28. Sonatype, Open Source Development and Application Security Survey 2014, bit.ly/2014OSS_survey
29. Wikipedia
30. Wikipedia
31. Wikipedia
32. Open Web Application Security Project, https://www.owasp.org/index.php/Top_10_2013-A9-Using_Components_with_Known_Vulnerabilities
33. FS-ISAC, http://bit.ly/fsisac_knownvulnerability
34. U.S. Representative Royce, http://royce.house.gov/news/documentsingle.aspx?DocumentID=397589
35. Toyota Supply Chain Management: A Strategic Approach to Toyota's Renowned System, by Ananth Iyer and Sridhar Seshadri